

WHAT IS CLAIMED IS:

1. A method of debugging a program in a computer system, comprising automatically removing at least a portion of all breakpoints associated with a particular job from the program when a debugger gets control of the program for the particular job.
2. The method of claim 1, wherein the portion is removed during a time when execution of the program is halted.
3. The method of claim 1, further comprising upon returning control from the debugger to the program, automatically reestablishing at least one useful breakpoint from the removed breakpoints.
4. The method of claim 3, wherein the at least one useful breakpoint is determined by executing a task to identify each unexecuted breakpoint located in an unexecuted portion of the program.
5. The method of claim 3, wherein the at least one useful breakpoint is determined by analyzing a behavior of a user responsible for setting the at least one useful breakpoint.
6. The method of claim 3, wherein the at least one useful breakpoint may be encountered by a thread which is allowed to execute when control is returned from the debugger to the program.
7. The method of claim 3, wherein upon returning control from the debugger to the program at least one thread is prevented from continuing execution and wherein any breakpoints which may have been encountered only by the at least one thread during continuing execution are considered useless and are not reestablished.
8. A method of debugging a program in a computer system, comprising:

upon a debugger getting control of the program for a particular job,
determining whether at least one useful breakpoint exists in the program for the
particular job;

if so, removing from the program all breakpoints associated with the
particular job except the at least one useful breakpoint;

prior to returning control to the program from the debugger, determining
whether at least one of the removed breakpoints is useful; and

if so, reestablishing the at least one of the removed breakpoints.

9. The method of claim 8, wherein at least one of (i) determining whether at
least one useful breakpoint exists in the program for the particular job and (ii)
determining whether at least one of the removed breakpoints is useful comprises
executing a task to identify at least one breakpoint which may be encountered upon
resuming execution of the program.

10. The method of claim 8, wherein at least one of the at least one useful
breakpoint and the at least one of the removed breakpoints is determined by
analyzing a behavior of a user responsible for setting the respective breakpoint.

11. The method of claim 8, further comprising returning control to the program.

12. A method of debugging a program in a multi-user computer system,
comprising:

during a time when execution of the program is halted, removing all
breakpoints associated with a particular job from the program;

executing a task to identify each useful breakpoint located in the program,
wherein a useful breakpoint is one which may be encountered upon resuming
execution of the program; and

if the task is completed before execution of the program is resumed, inserting
each useful breakpoint into its original location in the program.

13. The method of claim 12, wherein the multi-user computer system is a single

level store computer.

14. The method of claim 12, wherein the task is restarted in an event of removing and establishing at least one breakpoint.

15. The method of claim 12, further comprising inserting all removed breakpoints in the program if the task is not completed when execution of the program is resumed.

16. The method of claim 12, wherein the task is executed in response to hitting a breakpoint and wherein the task is restarted in an event of removing and establishing at least one breakpoint.

17. The method of claim 12, wherein executing the task comprises at least one of traversing a control flow graph of the program and traversing a call graph of the program.

18. The method of claim 12, wherein executing the task comprises:
 determining whether a call to a routine is made from a current routine at which execution halted; and
 if so, determining whether at least one of the routine and a called routine called from the routine contains a breakpoint;
 if so, adding the breakpoint to a useful breakpoint set.

19. The method of claim 18, wherein determining whether at least one of the routine and the called routine contains a breakpoint comprises:
 accessing a value in a call graph node of the routine, wherein the value is indicative of whether the at least one of the routine and the called routine contains the breakpoint;
 if the value indicates a presence of the breakpoint, traversing each node of a control flow graph, beginning with a node containing the routine.

20. The method of claim 19, wherein the task is performed for each routine of each thread of execution of the program.

21. A computer readable medium, comprising a program which, when executed by a processor in a multi-user system performs operations, comprising automatically removing at least a portion of all breakpoints associated with a particular job from the program when a debugger gets control of the program for the particular job.

22. The computer readable medium of claim 21, wherein the portion is removed during a time when execution of the program is halted.

23. The computer readable medium of claim 21, further comprising upon returning control from the debugger to the program, automatically reestablishing at least one useful breakpoint from the removed breakpoints.

24. The computer readable medium of claim 23, wherein the at least one useful breakpoint is determined by executing a task to identify each unexecuted breakpoint located in an unexecuted portion of the program.

25. The computer readable medium of claim 23, wherein the at least one useful breakpoint is determined by analyzing a behavior of a user responsible for setting the at least one useful breakpoint.

26. The computer readable medium of claim 23, wherein the at least one useful breakpoint may be encountered by a thread which is allowed to execute when control is returned from the debugger to the program.

27. The computer readable medium of claim 23, wherein upon returning control from the debugger to the program at least one thread is prevented from continuing execution and wherein any breakpoints which may have been encountered only by the at least one thread during continuing execution are considered useless and are not reestablished.

28. A computer readable medium, comprising a program which, when executed by a processor in a multi-user system performs operations, comprising:
during a time when execution of the program is halted, removing all breakpoints associated with a particular job from the program;
executing a task to identify each useful breakpoint located in the program, wherein a useful breakpoint is one which may be encountered upon resuming execution of the program; and
if the task is completed before execution of the program is resumed, inserting each useful breakpoint into its original location in the program.
29. The computer readable medium of claim 28, wherein the multi-user computer system is a single level store computer.
30. The computer readable medium of claim 28, wherein the task is restarted in an event of removing and establishing at least one breakpoint.
31. The computer readable medium of claim 28, further comprising inserting all removed breakpoints in the program if the task is not completed when execution of the program is resumed.
32. The computer readable medium of claim 28, wherein the task is executed in response to hitting a breakpoint and wherein the task is restarted in an event of removing and establishing at least one breakpoint.
33. The computer readable medium of claim 28, wherein executing the task comprises at least one of traversing a control flow graph of the program and traversing a call graph of the program.
34. The computer readable medium of claim 28, wherein executing the task comprises:
determining whether a call to a routine is made from a current routine at

which execution halted; and

if so, determining whether at least one of the routine and a called routine called from the routine contains a breakpoint;

if so, adding the breakpoint to a useful breakpoint set.

35. The computer readable medium of claim 34, wherein determining whether at least one of the routine and the called routine contains a breakpoint comprises:

accessing a value in a call graph node of the routine, wherein the value is indicative of whether the at least one of the routine and the called routine contains the breakpoint;

if the value indicates a presence of the breakpoint, traversing each node of a control flow graph, beginning with a node containing the routine.

36. The computer readable medium of claim 35, wherein the task is performed for each routine of each thread of execution of the program.

37. The computer readable medium of claim 28, wherein executing the task comprises, for each routine of each thread of execution of the program:

determining whether a current statement at which execution halted contains an unconditional breakpoint; and

if so, adding the unconditional breakpoint to a useful breakpoint set.

38. The computer readable medium of claim 37, further comprising, if the current statement at which execution halted does not contain an unconditional breakpoint:

determining whether a call to another routine is made; and

if so, determining whether at least one of the another routine and a called routine called from the another routine contains a breakpoint;

if so, adding the breakpoint to the useful breakpoint set.

39. The computer readable medium of claim 38, wherein determining whether the at least one of the another routine and the called routine contains the breakpoint comprises accessing a call graph node of at least the another routine.